

Éléments de correction sujet 07

Exercice 1

1

Avec cette méthode de mélange, on obtient :

['10', 'A', '9', 'R', '8', 'D', '7', 'V']

2

```
def liste_vers_pile(L):  
    N = len(L)  
    p_temp = Pile()  
    for i in range(N):  
        p_temp.empiler(L[i])  
    return p_temp
```

3

Il y a des erreurs dans l'énoncé pour la fonction *partage*, voici la fonction *partage* corrigée :

```
def partage(L):  
    N = len(L)  
    p_gauche = Pile()  
    p_droite = Pile()  
    for i in range(N//2):  
        p_gauche.empiler(L[i])  
    for i in range(N//2, N):  
        p_droite.empiler(L[i])  
    affichage_pile(p_gauche)  
    affichage_pile(p_droite)  
    return p_gauche, p_droite
```

On obtient l'affichage suivant :

3

2

1

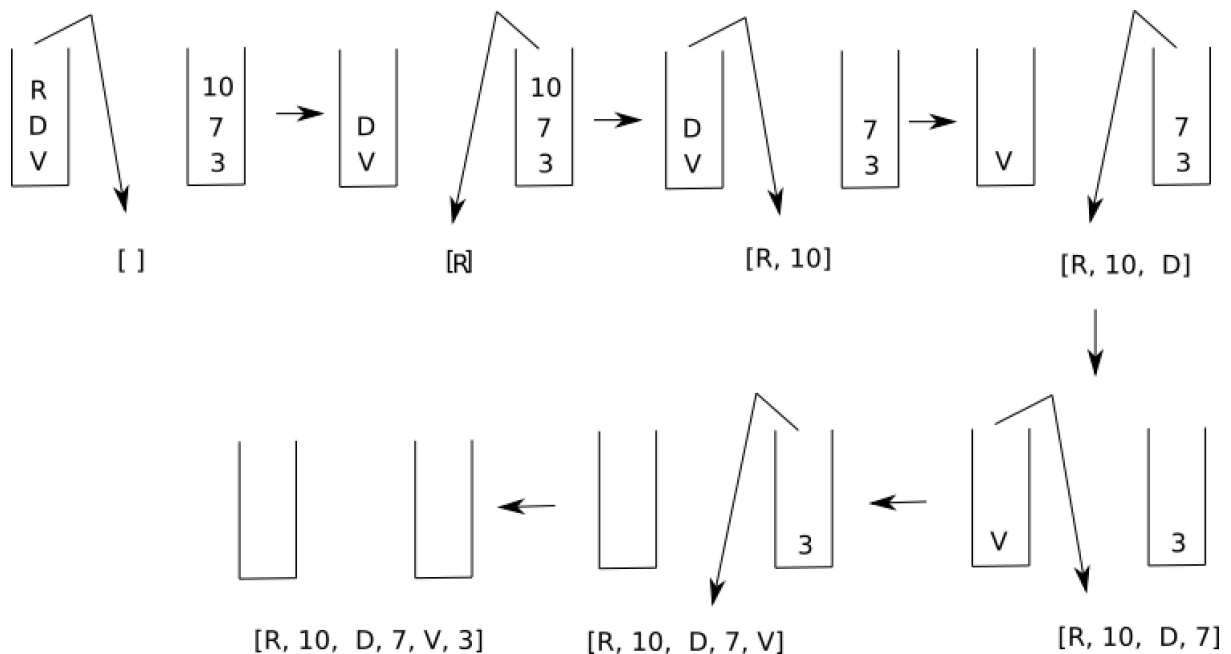
et

6

5

4

4a



4b

```
def fusion(p1, p2):
    L = []
    while not p1.est_vide():
        L.append(p1.depiler())
        L.append(p2.depiler())
    return L
```

5

Attention dans le sujet il y a une parenthèse en trop au niveau de 2e print

```
def affichage_pile(p):
    p_temp = p.copier()
    if p_temp.est_vide():
        print('_____')
    else:
        elt = p_temp.depiler()
        print('| ', elt, ' |')
        affichage_pile(p_temp)
```

Exercice 2

1

```
def mur(l,i,j):  
    return l[i][j]==1
```

2a

La variable d représente le carré de la distance entre les cases *case1* et *case2*. Deux cases adjacentes ont une distance égale à 1. Donc $d==1$ (et donc la fonction *voisine*) renvoie True si *case1* et *case2* sont adjacentes et False dans le cas contraire.

2b

```
def adjacentes(l):  
    adj = True  
    n = len(l)  
    for i in range(n-1):  
        if not voisine(l[i], l[i+1]):  
            adj = False  
    return adj
```

3

Avant le début de la boucle while la variable $i = 0$. À chaque itération de la boucle la variable i est incrémentée d'une unité ($i = i + 1$). La boucle va "s'arrêter" quand $i = \text{len}(\text{cases})$. Comme le tableau *cases* n'est pas infini, la boucle va donc obligatoirement se terminer.

4

```
def echappe(cases, laby):  
    nb_ligne = len(laby)  
    nb_colonne = len(laby[0])  
    return cases[0] == (0,0) and cases[len(cases)-1] == (nb_ligne-1,  
    nb_colonne-1) and teste(cases, laby)
```

Exercice 3

1

89_{10} correspond à 1011001_2

2

```
  11001110  
⊕ 01101011  
-----  
  10100101
```

3

```
def xor_crypt(message, cle):
    lst = []
    for i in range(len(message)):
        n = xor(ord(message[i]), ord(cle[i]))
        lst.append(n)
    return lst
```

4

```
def generer_cle(mot, n):
    cle = ""
    long_mot = len(mot)
    long_n = len(n)
    div = long_mot // long_n
    reste = long_mot % long_n
    for i in range(div):
        cle = cle + n
    for i in range(reste):
        cle = cle + n[i]
    return cle
```

5

E1	E2	E1 XOR E2	(E1 XOR E2) XOR E2
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

On remarque que (E1 XOR E2) XOR E2 correspond à E1.

Soit le message d'origine m, soit le message chiffré m' et soit la clé k.

Pour obtenir m' il faut faire un m XOR k.

Pour obtenir m à partir de m' et k, il suffit donc de faire un m' XOR k (m' correspond à "E1 XOR E2", m correspond à "E1" et k correspond à "E2").

Exercice 4

1a

L'attribut *nom* de la table *licenciers* ne peut pas servir de clé primaire, car il peut exister des homonymes et que la clé primaire doit être unique.

1b

L'attribut *id_licence* peut jouer le rôle de clé primaire.

2a

Cette requête renvoie le nom et le prénom des licenciés qui jouent dans l'équipe des "- 12 ans".

2b

En utilisant * à la place de prenom, nom, on obtient l'ensemble des attributs.

2c

```
SELECT date
FROM matchs
WHERE equipe = 'Vétérans' AND lieu = 'Domicile'
```

3

```
INSERT INTO licences
(id_licence, prenom, nom, annee_naissance, equipe)
VALUES
(287, 'Jean', 'Lavenue', 2001, 'Hommes 2')
```

4

```
UPDATE Licences
SET equipe = 'Vétérans'
WHERE prenom = 'Joseph' AND nom = 'Cuviller'
```

5

```
SELECT nom FROM licences
JOIN Matches ON licences.equipe = matches.equipe
WHERE adversaire = 'LSC' AND date = 19/06/2021
```

Exercice 5

1a

`Obj_bandeau.get_pixel_rgb(1)` renvoie le tuple (0, 0, 255) car LED1 est bleue.

1b

`Adafruit_WS2801.RGB_to_color(0, 0, 255)` renvoie l'entier 16711680 (*num_color* du bleu)

1c

Ces 2 instructions permettent d'afficher le code de la couleur la led LED0, c'est-à-dire 255.

2a

BLEU	BLEU	BLEU	BLEU	BLEU	BLANC	BLANC	BLANC	BLANC	BLANC	ROUGE	ROUGE	ROUGE	ROUGE	ROUGE
------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

2b

VERT	JAUNE	JAUNE	VERT	JAUNE	JAUNE	VERT	JAUNE	JAUNE	VERT	JAUNE	JAUNE	VERT	JAUNE	JAUNE
------	-------	-------	------	-------	-------	------	-------	-------	------	-------	-------	------	-------	-------

3a

La méthode `__init__` permet d'initialiser une instance de la classe `Bandeau`. Elle prend en paramètre le nombre de leds utilisées dans le bandeau.

3b

Permet de faire passer les leds 6 et 7 au bleu.